Placement of Tab for Text Compression and Auto Text Formatting Documentation:

The Tab title included should be inserted into the tab and located before Section LI

VALPAR INTERNATIONAL
3801 E. 34TH STREET
TUCSON, ARIZONA 85713
602-790-7141

valFORTH T.M.

SOFTWARE SYSTEM
for ATARI*

Text Compression and Auto Text Formatting

valFORTH™
SOFTWARE SYSTEM

Text Compression and Auto Text Formatting

Evan Rosen

VALPAR INTERNATIONAL

Disclaimer of Warranty
on Computer Programs

All Valpar International computer programs are distributed on an "as is" basis without warranty of any kind. The total risk as to the quality and performance of such programs is with the purchaser. Should the programs prove defective following their purchase, the purchaser and not the manufacturer, distributor, or retailer assumes the entire cost of all necessary servicing or repair

Valpar International shall have no liability or responsibility to a purchaser, customer, or any other person or entity with respect to any liability, loss, or damage caused directly or indirectly by computer programs sold by Valpar International. This disclaimer includes but is not limited to any interruption of service, loss of business or anticipatory profits or consequential damages resulting from the use or operation of such computer programs.

Defective media (diskettes) will be replaced if diskette(s) is returned to Valpar International within 30 days of date of sale to user.

Defective media (diskettes) which is returned after the 30 day sale date will be replaced upon the receipt by Valpar of a $12.00 Replacement Fee.

TEXT COMPRESSION AND AUTO TEXT FORMATTING

Table of Contents

NOTICE
TEXT COMPRESSION AND AUTOMATIC TEXT FORMATTING
CODE TRANSPORTATION

The routines in this package have been coded and presented so that they may be readily transported to other fig-FORTH systems on machines other than the Atari 400/800. This is in response to numerous requests to this effect from various "adventure" game authors. We note, however, that the same restrictions apply to the software in this package, whether run on the Atari 400/800 machines or any other:

First, the code may ONLY be used in either an AUTO'd system as described in valFORTH 1.1 documentation, or in a target-compiled system.

Second, any software written with these routines, on any machine, must contain the acknowledgement of Valpar International as the source of the code, as described and detailed in valFORTH 1.1 documentation.

Other distribution may be construed to be a violation of applicable copyright laws.

Overview

This package attempts to fill at least two common needs of the programmer who does verbal/interactive programming.

First, a group of automatic text formatting routines is provided that allow two different approaches:

*A non-wrap line formatter to both the video display and the printer, including variable-margin capability and inverse video option, and

*A versatile window formatting system, with scrolling, color and inverse video options as appropriate, and window naming. Notes on the creation of window types with different "generic" parameters are also included.

In both modes described above, user options of left-, right-, center-, or fill-justification are supported, as is numerical output formatting.

Second, two different approaches to the problem loosely termed "text compression" are implemented:

The first is intended for use in programs where run-time retrieval of text stored on disk is allowable, and provides a set of general virtual-memory operators for the creation and retrieval of messages from disk. A simple encryption scheme is provided as a (working) example for the software developer who wishes his or her messages to be not easily readable from disk with, for instance, a Forth screen editor. In addition, alternate points in the virtual memory routines are indicated where deeper encryption routines might be employed. Routines are provided for virtual memory message programming on both one- and two-drive development systems.

The second set of text compression routines is intended for use in "in memory" applications, such as cassette-loaded programs, that do not have access to disk for message retrieval. In this case the most compact code practical is desired, and a system built around some of the basic aspects of Forth's compact threaded-code structure is provided.

Finally, we note that the autoformatting and text compression utilities are designed to be used in most any of their possibly different combinations.

STROLLING THROUGH TCAF
(Text Compression and Autoformatting)

The organization of this disk is slightly different from the others in this series. While a table of contents may still be found on screen 170 as usual, in this package the "load chain," starting on screen 166, will get far more use. In general when one wishes to load a TCAF development system with a specific set of capabilities, one makes slight adjustments to the load chain option screen and then simply loads the first screen in the chain. The chain does the rest.

To start off, first prepare two blank, formatted disks. Make your normal working copy of TCAF on the first disk and leave it un-write-protected. The second disk will be used a little later.

Autoformatting

In order to select options you will want to make changes to the load chain option screen. This may be found by locating the load chain in the directory on screen 170, and then scanning through the screens in the chain until you find the one marked "options" in its first line. (This is on or near screen 167.) Look at this screen, and see that most of the lines have a left parenthesis in the left column, followed by a LOAD command and a comment. By removing selected left-column left parentheses you can activate various options. Right now on your working copy use an editor to remove all of the left-column left parentheses except for the one on the line that says "text compression." (Text compression uses transient structures and will be discussed separately.) And, of course, don't remove the one in the comment at the very top of the screen. OK, now boot a bare valFORTH 1.1 system, and load in the debugger, and swap in the TCAF disk, do MTB as usual, and load the first screen in the load chain on this disk. (Probably 166.)

When the prompt comes back, type

```
     ON STACK
```

since you'll want to watch the stack. Then type

```
     TYPEOUT
```

(Failure to execute this initialization word may cause a crash as you try to use words like *TYPE later on.) This command activates one of the two formatting modes. This mode, called "type-out mode," since it uses the word TYPE as its actual output word, can send formatted type to either the display or the printer. The other formatting mode is activated by WINDOUT and is called "window-out mode." It will be discussed a bit later.

Now type

```
     " Here is a simple example of the formatter's function."
```

using lower case as shown, and notice that an address, actually PAD, is left on the stack. Now reactivate upper case (press Shift and Caps-Lowr) if you haven't already, and type

```
     COUNT
```

The address was bumped by one, and the string count was extracted from the first byte in the string created by " and placed on top of stack. All normal. Now type

```
     2DUP CR CR TYPE CR
```

and see that the typed output wraps around as usual. Now try

```
     2DUP CR CR *TYPE *CR
```

The word "formatter's" is no longer split. Let's try it again but with different formatting. Type

```
     CTRJST  ("center justification")
     2DUP CR CR *TYPE *CR
```

How about

```
     FILJST  ("fill justification")
     2DUP CR CR *TYPE *CR
```

The text is now spread or "filled" to take up the whole space between the margins. The last mode is

```
     RGTJST  ("right justification")
     2DUP CR CR *TYPE *CR
```

which gives the expected result. Finally, type

```
     LFTJST  ("left justification," the default mode)
     2DUP CR CR *TYPE *CR
```

and we're back where we started.

Well, what precisely is happening? The 2DUP each time is there of course to reproduce the two stack arguments, adress and count, for use by *TYPE (or TYPE). The two CR's each time are merely to space the result down the page a bit, and make it start at the left margin. As we will see, these two CR's will not generally be necessary in normal programs. The TYPE we'll assume you already know about. If not, look it up in the 1.1 glossary. While you're looking at TYPE's definition you might refresh your memory about how to allow it to type inverse video characters also. A short discussion about this follows TYPE's definition, and we may need this feature later on. Oh, what about *TYPE? *TYPE, like TYPE, takes a count and address on the stack, but instead of routing the text directly to an output device, *TYPE sends it instead to a holding buffer, located at BUF, where it accumulates. As each character is sent to the buffer it may be colored, inversed, or capitalized, depending on whether these options are loaded and appropriate. Since we loaded all three of these options we'll try them presently. When the buffer at BUF overflows with a non-blank character, *TYPE formats the line (if any format routines were loaded) and then sends it out via a vect called *XMTLN. Roughly, a vect is a word that can be "assigned" the meaning of a second word so that when the vect is executed it acts precisely like the word last assigned to it.) "XMTLN" in *XMTLN stands for "transmit line." The word *XMTLNP is currently assigned to *XMTLN and is located, in the first release, on or near screen 73. *XMTLNP, and so now *XMTLN, types out the buffer at BUF and increments a line counter if the printer is on and does a few CR's if a printed page is full. After the buffer is output, cleared, and the overhanging characters have been moved to the beginning of the buffer, *TYPE continues to consume the character string. In general there will always be something in the buffer unless it has been cleared. *CR pushes the last of the text out of the buffer to the output device, and then clears and initializes the buffer with BUFINIT. You probably won't have to do BUFINIT yourself unless you are experimenting with the internals of the program. To illustrate this point about *CR, type

```
     2DUP  CR CR *TYPE 2DUP *TYPE *CR
```

This time, since we didn't do *CR after the first *TYPE, the next *TYPE tacked its text right on to what was left in the buffer.

Now type

```
     SP!  (we'll make a new message)
     : SHOW 2DUP CR CR *TYPE *CR ;
     " here is another message for another purpose."
     COUNT
     CTRJST
     CAP SHOW
     SHOW
```

See what CAP does? Now try

```
     ON CAPS
     SHOW
     OFF CAPS
     SHOW
```

And what about inverse video? Since *TYPE uses TYPE and TYPE as it now stands will not print inverse video (it strips the high bit before sending a byte out) we'll need the modification discussed in the 1.3 Glossary, under TYPE. Here it is, type it in, c a r e f u l l y:

```
     HEX  FF  '  TYPE  14 + C!  DECIMAL
```

and then type

```
     VLIST
```

to see an interesting side note. The high bits of the last byte of (almost) all names are set. This is used by vocabulary search words. But, back to business. Type

```
     SHOW
```

And you get trash. This is because some word in VLIST uses PAD for something, so your message was overwritten. This is just a reminder. Type

```
     SP! (clear the stack)
     " Here is yet another message, to show other features."
     COUNT
     ON CAPS
     ON INVID
     SHOW
     ON INVBK  (for "inverse background")
     SHOW
     OFF INVID
     SHOW
     2DUP CR *CR *TYPE *CR *CR CR
     etc.
```

Play with these things for a while if you like. When you're done, do

```
     SP!
     OFF CAPS
     OFF INVID
     OFF INVBK
```

and we'll continue.

Virtual Memory (Disk-stored) Text

We used the word " in the exercises above, but it has a serious shortcoming in that it won't digest a string larger than 255 characters since it only keeps a one-byte length byte. The word X" (for "extended quote") in this package allows longer strings when loading from disk. X" will not work from the key-board. Use XCOUNT with X", and it retrieves the two-byte length count and leaves it on top of the stack. There is a demo of X" on screen 120. Take a look at it if you like and then type

```
     120 LOAD
```

and a short message will come back as part of the demo. Turn your stack on if you've turned it off for any reason, and do

```
     XCOUNT
     OFF CAPS  OFF INVID  OFF INVBK
     FILJST
     SHOW
```

If you looked at the screen you may have noticed the right-arrow characters near the end of the text. These cause a *CR to be executed at that point in the text. See *EMIT code for details. You can make your own control characters in a similar fashion. (To type a right arrow character in the valFORTH 1.1 editor, do ESC followed by CTRL-*). Observe also that no --> was required for X" to cross the screen boundary. X" will only stop on finding a final " and so may run right on through a disk looking for one if you forget to put it in.

Well, X" is ok, but not as handy as it might be for general programming. Look at screen 122 and then type

```
     SP!
     122 LOAD
```

and a demo message will come back again, indicating that a new word, MSGDEM1, now exists. This word will actually pull its message text off the disk. Let's do it. Do MTB just to make sure it's not cheating, and then type

```
     CR CR MSGDEM1
```

Notice that we didn't use SHOW this time, just the message name. The messages end with a right-arrow. Now, the method that generates this message, namely using a new word. V" , followed by a string and then a terminating " and then the word M: followed by the message name, does achieve the desired result, but at the price of leaving the V", ", M:, and name on the disk along with the message. This method is provided only because for those working with a one-drive development system it is the easiest, and does not involve any disk swapping during compile time. However, for those with two drive systems, and those with only one drive but also a tolerance for swapping disks every time a message is compiled, the next and last structure in this series is provided. It allows fully compact, text-only messages to be compiled on the final product, and also allows encryption of the text. We will first do it the way the one-drivers need to.

Look at screen 124. The 80 ALTINIT command sets up an alternate set of disk pointers to start at screen 80. This is where, in our example, the text of the various messages compiled by this method will be stored on the extra disk we formatted at the beginning. Notice that the message starts with X" again. Hence, we see that it will first be assembled at PAD before being sent elsewhere. Now look at screen 125. There's the terminating " , the defining word, MSG:, and the message name, and a short message with ." This final message is just there for convenience in this demo and is not needed in general. Type

```
     124 LOAD
```

and when it tells you to put in the destination disk, swap in the extra blank disk you formatted, then press START as directed. At the next prompt, swap back and press START again. When using this method you must be very careful not to reverse your disks or you may wipe out part of your source disk.

Well, the message is now written to the second disk on screen 80, and the word MSGDEM2 knows where to find it. Let's take a quick look to see that it's really there. Type

```
     MTB
```

to empty the buffers, and then swap disks again (so that the destination disk is in the drive) and do 80 LIST, and then 81 LIST. There's the message. The first two strange bytes on screen 80 are the count. Now do

```
     MTB  CR CR  MSGDEM2
```

and watch the routines pull the message from the disk. While we're here, let's send this to the printer. But since your printer may have characteristics different from the printer this package is initialized for, we want to adjust a couple of things. The first item is a quan named PWID. This is the actual number of columns your printer has. The default value is 80. To change it to 96, for example, type

```
     96 TO PWID
```

The second item is the quan PRTWID which is the width of the area you'd like to print to. The default again is 80. To set it to 60, say, type

```
     60 TO PRTWID
```

The third item is how far you'd like to indent. This is the quan PRTIND and its initial value is 0. To set it to 10 type

```
     10 TO PRTIND
```

Finally, we want to tell the formatter to send its output to the printer now, so type

```
     PRT:
```

(The default setting was to the video display, and will be called back by VID:) Is your printer ready? Let's try it. Type

```
     MSGDEM2
```

Since many printers will get confused if a character with the high bit set is sent to them you might want to be careful about this.

Incidentally, the same options are available with the video display. PRTWID becomes VIDWID and PRTIND becomes VIDIND. PRT: becomes VID:. There is no "VWID" since the formatter derives this from the positioning of the margins. (The left margin is kept by the OS in the byte at 82 decimal, and the right margin byte is at 83. Default are 2 and 39 respectively.)

Any new printer settings only become active when PRT: is executed, and likewise with video settings and VID:.

Try

```
     20 TO VIDWID
     4 TO VIDIND
     VID:
     CR CR MSGDEM2
     38 TO VIDWID  (back to default)
     0  TO VIDIND  (ditto)
     VID:   (move in new values)
```

OK, now swap the source disk back in, that is, the TCAF working disk, but keep the destination disk handy. Let's load a few (six) more messages. Type

```
     MTB (to empty the buffers)
     126 LOAD
```

and follow the prompts.

As you can see, any large amount of this single-drive compilation could be quite tiresome. Do a short VLIST (abort with any of the three yellow console buttons) and look at the new messages. Swap in the destination disk, do MTB, and then try

```
     CR M0
     CR M1
     etc.
```

Encryption and 2-Drive Systems

There are two more features to point out. They are encryption/decryption (e/d) and adjustments for two-drive systems. Concerning e/d, look at screen 105, or wherever you find the title EN, DECRYPT or similar. (Do an INDEX if you can't find the right screen easily.) Notice that there is a --> at the top of this screen which is causing it not to load. Remove this arrow with your editor. (Since you're going to reload the system in a minute anyway, it's ok if you over-write the system to get an editor in. Get one in somehow.) Now on the next two screens you should find the words ENCRYPT and DECRYPT in parentheses. (DECRYPT is in three times.) Remove the parens to allow these two words to load. Now, you folks with two drives, find the screen where MSG: is defined. (On or near screen 112). There are several sets of parens. Leave the ones that enclose $ENCRYPT and ... $DECRYPT ... alone. Shift only the ones that are around " DR1 or " to be around " or DSTDSK " and shift the ones around " DR0 or " to be around " or SRCDSK ." Just to be on the safe side, here's a picture of how the screens should look after these changes.

FOR ONE OR TWO DRIVE SYSTEMS

```
Scr # 105
( Vrtxt:  EN,DECRYPT  example  )



: ENCRYPT                ( c1 -- c2 )
   117 - DUP 0<
   IF 256 + ENDIF ;

: DECRYPT                ( c2 -- c1 )
   117 + DUP 255 >
   IF 256 - ENDIF ;





                                   -->
```

```
Scr # 106
( Vrtxt:  V$TP  V$@  V$*EMT     )

: V$TP              ( -- X$ [=PAD] )
  VRTC@   DECRYPT   NXTVRT VRTC@
    DECRYPT   NXTVRT 256 * + ;

: V$@               ( -- X$ [=PAD] )
  PAD 2+ V$TP DUP PAD ! 0
  DO VRTC@ OVER C! 1+ NXTVRT
  LOOP DROP PAD ;

: V$*EMT            ( -- X$ [=PAD] )
  V$TP 0
  DO VRTC@   DECRYPT
     *EMIT NXTVRT
  LOOP ;                           -->
```

```
Scr # 107
( Vrtxt:  V$!                   )

: V$!                      ( X$ -- )
  DUP @ 2+ 0
  DO DUP C@   ENCRYPT
     VRTC! 1+ NXTVRT
  LOOP DROP ;









                                   -->
```

FOR TWO DRIVE SYSTEMS ONLY

```
Scr # 112
( Vrtxt:  ALT$!  MSG:                )

: ALT$!                    ( X$ -- )
  VRTSAV ALTREC V$!
  ALTSAV VRTREC ;

: MSG:                     ( X$ -- )
  ( *ENCRYPT )
  <BUILDS   DR1 ( or   DSTDSK )
   ALTBLK , ALTIN , ALT$!
   FLUSH    DR0 ( or   SRCDSK )
  DOES> VRTSAV
   DUP @ SWAP 2+ @ IN ! BLK !
   V$*EMT ( or )
   ( V$@ *DECRYPT XCOUNT *TYPE )
   VRTREC ;
```

What this last change does is substitute an automatic disk-shift for the swap prompts driven by DSTDSK and SRCDSK. These words are no longer needed and may be bypassed later on, if you're comfortable with the way things are working. OK, now, although some of you might be able to get these changes in by doing some FORGETting and reloading (if you didn't have to overwrite the system before), why not just reload the whole (modified) system this time, starting from valFORTH 1.1. Don't forget the debugger if you want it, and remember to initialize with TYPEOUT. Go ahead. We'll wait.

Now you can repeat the exercises from before, starting where you did the 174 LOAD. Two drive systems should have the "destination" disk, the one with the messages on it, in the second drive, and the source disk, TCAF, in the first drive. Two-drive systems should execute DR1 (which in FORTH means the second drive, which is number 2 on Atari systems) before saying message names so that the code will read the right drive. Say DR0 to go back to the source-code drive. (Released programs will of course not want to say DR1 since they will expect the "game" or application disk to be in the first drive, DR0, which is default. When you are making messages in this way, be sure to start them high enough on disk that your AUTO'd program, which will actually do the message retrieval, will fit under them.) If you look at screen 80 on the destination disk, what you'll see is that the text is now scrambled. The encryption routine used is a simple off-set scheme, and would be easy to crack for a serious hobbyist, though not for the casual user. If you are interested in a higher degree of security, you can encode a whole string at a time with more sophisticated routines. A pair of names, $ENCRYPT and $DECRYPT, have been reserved for these routines. We don't provide any examples, but a modern text on cryptography might be a good place to start looking. Anyway, if you use the names $ENCRYPT and $DECRYPT, and if the routines expect an extended string on stack (that is, one with a two-byte count at its front end) then they will (hopefully) snap right into the spot designated by the parens.

Windows

Windows are rectangular areas of the video display. They are not supported on the printer, but are supported in both "black and white" graphics 0 mode, and colored graphics 1 and 2 modes. Windows may be set up on-the-fly or they may be given names so that words can call them up readily. The implementation provided here may be used as an example and guide, since you may want your windows to act somewhat differently.

Since it is tricky to interact with a graphics 0 window from the keyboard (there is no simple way that we can find to date to create a split-screen option) we'll illustrate windows in graphics 1, and so also show how color works. Type

```
     1 GR.
     3 4 10 5 MAKECW
```

This makes a Color Window whose upper left hand corner is at the 3rd column over, 4th row down (counting the left and top edges as zero), and which is

10 characters wide and 5 high. We have messages M0 through M5 still available, so let's send them to the window. Type

```
     WINDOUT  (counterpart of TYPEOUT)
     OFF CAPS
     0 COLOR M0
     1 COLOR M1
     2 COLOR M2
     3 COLOR M3
     WCLR
```

Note the extra coloration caused by the mix of upper and lower clase. This can be canceled by ON CAPS though it restricts the user to two color for the letters instead of four. A good practice would be to put all the source text for colored windows in upper case. Coloration switches in the middle of a message could be implemented by control characters similar to the right-arrow character and its meaning of *CR. This is done in *EMIT, you'll remember, and you might even use a case statement.

Numerical formatting is also supported, by the words *. and *.R which are direct counterparts of . and .R, except that they go through the formatter before outputting. Try

```
     3456  *. *CR
     7890 7 *.R *CR
```

These routines should be used both with WINDOUT and TYPEOUT. Using just . or .R will upset the formatting.

There is also

```
     2 2 15 5 NAMECW  BINGO
```

which names a color window with the given parameters as BINGO. When BINGO is executed it will clear itself and position the imaginary cursor at its the upper lefthand corner. By studying the code, this and other performance characteristics may be altered.

Text Compression

Finally, there is "true" text compression (TC) itself. TC is intended primarily for applications where disk access to messages is not available, such as in cassette-hosted systems. This utility uses transient structures which you have probably come across before in the packages in this series. Hence, all the warnings about memory collisions must to some extent apply. The text compression utilities themselves are fairly straightforward to use, but what they do is rather complex. Briefly, TC allows the creation of bits of headerless code called "tc-texts" that, when executed, put a string onto the stack and then jump to the appropriate Forth words, for formatting. These tc-texts come in three types in this package, namely, tc-words, tc-suffixes and tc-prefixes.

The general procedure is to:

(1) Load the text compression routines.
(2) Define all needed tc-texts.
(3) Define all words that use tc-texts within themselves.
(4) Execute DISPOSE which will sever links to all of the tc-text creating and compiling structures, leaving only minimal, minimal, headerless structures.

Since (it turns out) we can load the text-compression routines right on top of the rest of the code we already have in, let's do that. Look at the screen in the load chain which you modified at the beginning of this excursion. It was probably 167. Text compression was not loaded at that time. Note the load screen for text compression (probably screen 60) and load it. Because this section uses transients you cannot use SAVE to create a bootable copy until you have executed DISPOSE to break the links to the transient area. In addition, FORGET will act a bit odd, and may cause crashes, so try to avoid using it until you have disposed. There are only three new words to learn in text compression, namely, W= , P= , and S=. These words define tc-words, tc-prefixes and tc-suffixes. For example,

```
W= DOG
W= TAIL
P= SUPER
S= 'S
S= !
```

defines five tc-texts. Type in the five definitions above and then type

```
DOG DOG DOG DOG DOG CR *CR
SUPER DOG CR *CR
SUPER DOG 'S TAIL ! CR *CR
```

The justification, capitalization, coloring, window output, and other options will also function with tc-texts.

Obviously, there is potential for numerous word-name conflicts between tc-texts and FORTH. The punctuation marks, for instance P= . P= , P= ! and so on all are desirable and all already exist in the FORTH vocabulary. Hence the three defining words for tc-texts automatically put the words they define into a separate vocabulary named ^^. In addition, the name | (Shift-+) has been assigned as an alias for FORTH to shorten source code and ease typing. For instance, one might have a FORTH word like:

```
: ?TL                              ( flag -- )
  IF  ^  A_DOG_'S_TAIL_IS_HERE !  *CR  |
  ENDIF ;
```

By going into the ^ vocabulary the tc-text ! was interpreted properly, instead of as the FORTH !. Similarly, by going back into the | (FORTH) vocabulary, the word ; was interpreted as the FORTH ; rather than as some prefix that might have been in the ^^ vocabulary.

TEXT COMPRESSION AND AUTO TEXT FORMATTING GLOSSARY

<u>Basic Commands</u>

```
*."                     ( -- )
```

Like ." , but sends string to the active formatting/outputting routines.

```
*TYPE                   ( addr count -- )
```

Like TYPE, but sends string of count characters starting at addr to the active formatting/outputting routines.

```
*CR                     ( -- )
```

Somewhat like CR in that it causes a carriage return. In addition, *CR first formats and flushes the buffer to the output device, and clears the buffer after doing so.

```
*EMIT                   ( c -- )
```

Like EMIT except sends the character c to the formatter, instead of directly to the output device.

```
*SPACE                  ( -- )
```

Sends a single character of value in the quan BKGND to the formatter, through *EMIT.

```
*SPACES                 ( n -- )
```

Sends n characters of value in the quan BKGND to the formatter, through *EMIT.

```
*BACKS                  ( -- )
```

Similar to action of delete key. Backs up the formatter buffer pointer, BPTR, one location and fills new location with BKGND value.

```
RGTJST                  ( -- )
```

Sets up formatter for right justification.

```
LFTJST                  ( -- )
```

Sets up formatter for left justification.

```
CTRJST                  ( -- )
```

Sets up formatter for center justification.

FILJST (--)

Sets up formatter for fill justification.

INVID (f --)

ON INVID means text will be output in inverse video; OFF INVID means normal video.

INVBK (f --)

ON INVBK means background of text will be output in inverse video. OFF INVID means normal video.

CAP (--)

Causes capitalization of the next byte processed by *EMIT or *TYPE.

CAPS (f --)

ON CAPS means subsequent formatted text will be capitalized if lower case. OFF CAPS means text will be printed as-is.

COLOR (b --)

New color register b will be used for color of subsequent text output to windows in Graphics modes 1 and 2.

TYPEOUT (--)

Initialization routine for the formatter. Either TYPEOUT or WINDOUT must be executed before the first attempt to output text from the formatter or the system may crash. TYPEOUT directs the formatter to use TYPE as its actual output routine, allowing output to the display screen or printer.

WINDOUT (--)

Initialization routine for the formatter. Either TYPEOUT or WINDOUT must be executed before the first attempt to output text from the formatter or the system may crash. WINDOUT directs the formatter to use window routines for output. A window must be created before attempting to use window output or the system may crash. See also NAMWND.

Several more points are worth noting:

* If you are programming short phrases that do not generally run together, you can save some memory by defining a <BUILDS DOES> construct that always attaches the *CR to the end of the operation, thus saving two bytes per message, with the new <BUILD DOES> that loaded with this package.

* If you want to create new types of tc-texts, such as one to deal with problems like SHINE ING, just follow the examples of how the words W= P= S= are constructed. Smart prefixes that strip trailing vowels, for example, would not be difficult to code, but would not necessarily be worth the memory cost. However, in a very large application it might well be worth coding a large number of spelling rules.

* To create tc-texts that contain blanks, create a control character that is not printed, and use this as the blank. The character we suggest for this is the underline, whose ATASCII is 95. Note how the right-arrow, ATASCII 31, is picked off by *EMIT. Do the same for 95, only make it perform *SPACE instead of *CR as right-arrow does.

* Some tc-texts will get rather long, and will be cumbersome in source text. They can be provided with a no-cost alias. For example, say we had

```
     W= A_DOG_WITH_A_BONE
```

We could then add

```
     TRANSIENT
     : D&B ^ [COMPILE] A_DOG_WITH_A_BONE_ ; IMMEDIATE
     PERMANENT
```

This alias would be removed by DISPOSE, as would, of course, the tc-text name A_DOG_WITH_A_BONE, leaving only the headerless tc-text itself.

* As set up, the Transient system is 4000 bytes below the display list. This may not be enough for some applications. The way to find out how much room you have left in the transient area is to type

```
        TRANSIENT 741 @ HERE - U. PERMANENT
```

You might even define a word to do this. Call it TFREE. A trap in CREATE, and so also in : is designed to keep you from actually running into the display list by simply aborting the definition in progress when there are less than 128 bytes left.

* Finally, always remember to DISPOSE when you're done with the transients. If you forget and do SAVE you will not get a working system.

This system of compression is quite compact, costing only two bytes to produce output from a tc-text. The cost in memory of producing the tc-text of a word of n letters, (not even counting the trailing blank) is only n + 2.

Quans, vects, and subcommands

FDIR (-- +-1)

A quan that holds the next direction to be used by the fill-justification routines when padding the text in the formatting buffer with blanks.

*JUST (--)

A vect used to point to the routine that performs whatever justification action is current. Altered by LFTJST, RGTJST, CTRJST, and FILJST.

BKGND (-- n)

Quan which holds the value of the background character to be used when clearing the formatting buffer. Generally either 32 (blank) or 160 (inverse blank.) See INVBK.

EOB (-- n)

Quan which points to the location in the formatter buffer corresponding to the last allowable position in the current output width. Set up by various routines including PRT:, VID:, and window-creating routines. Stands for "end of buffer."

BPTR (-- n)

Quan which points to the next available location in the formatter buffer. May be user-altered for special purposes, but should not be placed lower than BUF or higher than EOB. Stands for "buffer pointer."

WWID (-- n)

Quan which holds width of field to which text will be output. Used to set up EOB, which is actually used by the formatting routines See EOB. Stands for "window width" though windows as defined elsewhere need not exist.

*XMTLN (--)

A vect that points to the routine to be used to move text from the formatter buffer to the output device. Set up at present either by TYPEOUT or WINDOUT. Stands for "transmit line."

BUF (--)

A label that points to the beginning of the formatter buffer area. This area need only be three bytes longer than the longest line to be formatted.

INVBK (ON or OFF --)

When ON, background character output by formatter in D graphics mode will be inverse video blank. When OFF, this character will be normal video blank. Sets up BKGND. See BKGND.

BUFCLR (--)

Fills the formatter buffer with BKGND.

BUFINIT (--)

Fill the formatter buffer with BKGND, sets up EOB using WWID and BUF, and points sets BPTR equal to BUF.

*TINT (c -- c)

A vect that either points to the coloring routines when a color window is active, or to NOOP when a 0 graphics window is active.

*CAP (c -- c)

Capitalization routine.

*INV (c -- c)

A vect that either points to the inversing routine when a 0 graphics window is active, or to NOOP when a color window is active.

Text Compression

```
W= xxx, ( -- )
   xxx: ( -- )
```

Creates a tc-word-compiling word, named xxx, and a headerless tc-word which when executed sends the string xxx through the formatter followed by *SPACE. xxx when executed, compiles in the cfa of this tc-word. W= and xxx are both in transient area and so are disposed by DISPOSE.

```
P= xxx; ( -- )
   xxx, ( -- )
```

Creates a tc-prefix-compiling word, named xxx, and a headerless tc-prefix which when executed sends the string xxx through the formatter. xxx when executed, compiles in the cfa of this tc-prefix. P= and xxx are both in the transient area and so are disposed by DISPOSE.

```
S= xxx, ( -- )
   xxx, ( -- )
```

Creates a tc-suffix-compiling word, named xxx and a headerless tc-suffix which when executed sends the string xxx through the formatter preceded by *BACKS and followed by *SPACE. xxx, when executed, compiles in the cfa of this tc-suffix. S= and xxx are both in the transient area and so are disposed by DISPOSE.

Typed Output

PRTWID (-- n)

A quan containing the width of the area to be printed when printer output from the formatter has been selected by PRT:. PRT:, among other things, moves PRTWID to WWID.

PRTIND (-- n)

A quan containing the number of spaces the printer is to indent when outputting from the formatter. PRTIND is moved to PVIND by PRT:

PVIND (-- n)

A quan containing the number of spaces the output device is to indent when outputting from the formatter. Set up by PRT: from PRTIND or by VID: from VIDIND.

PWID (-- n)

A quan containing the number of columns the printer is actually able to print as it is currently configured, and independent of the formatting routines.

VIDIND (-- n)

A quan containing the number of spaces the output routines is to indent when outputting from the formatter. VIDIND is moved into PVIND by VID:.

VIDWID (-- n)

A quan containing the width of the area to be written when video output from the formatter has been selected by VID:. VID:, among other things, moves VIDWID to WWID.

PRT: (--)

Directs TYPEd output to the printer, and moves appropriate values into WWID and PVIND.

VID: (--)

Directs TYPEd output to the video display, and moves appropriate values into WWID and PVIND.

PRINIT (--)

Resets PCTR, the printed line counter.

*XMTLNP (--)

Routine sent to the vect *XMTLN by TYPEOUT. Routes output through TYPE.

Windows

WADR (--)

Address in memory corresponding to character position in upper lefthand corner of current window.

WHGT (--)

Height in lines of currently active window.

LPTR (--)

Counter that holds number of next line in window to which text is to be written. If LPTR points beyond the window then scrolling will occur at next output.

B/LN (-- n)

Bytes per line. Necessary datum for scrolling and clearing routines for windows.

WCLR (--)

Fills the current window with BKGND.

NAMWND (wadr wid hght b/ch byt/ln --)

One of many possible window-defining structures. Accepts window upper lefthand corner address, its width, height, byte-character, and the bytes/ln of the current graphics mode.

```
NAMEBW   xxx, ( column row wid hgt -- )
         xxx: ( -- )
```

Names a 0 graphics window for later activation.

MAKEBW (col row wid hgt --)

Establishes a 0 graphics window immediately but does not name it for later retrieval.

```
NAMECW   xxx, ( col row wid hgt -- )
         xxx: ( -- )
```

Names a 1 or 2 graphics window for later activation.

MAKECW (col row wid hgt --)

Establishes a 0 graphics window immediately but does not name it for later retrieval.

Virtual (Disk-based) Memory

(A pointer to a byte on disk is implemented by the two system variables, BLK and IN in the fig model. BLK contains the block number pointed to and IN contains the number of bytes into the block the byte in question is located.)

VRTC@ (-- b)

Fetches the byte pointed to on disk by the system variable BLK and IN. (BLK is the block number, and IN is the number of bytes into the block the desired byte is located.)

VRTC! (b --)

Stores the byte on stack to the location on disk pointed to by BLK and IN. See VRTC@.

VRTSAV (--)

Saves the values of system variables BLK and IN to quans OBLK and OIN respectively.

VRTREC (--)

Recalls the values of the system variables BLK and IN from the quans OBLK and OIN respectively.

NXTVRT (--)

Bumps the system variables BLK and IN as required to point to the next location in virtual memory.

RELVRT (offset --)

Takes an offset on stack and alters the system variables BLK and IN as necessary to point offset bytes from their initial virtual memory location.

V" (-- blk in)

Leaves the values of BLK adn IN on the stack at the time it is executed and then scans the virtual memory pointer formed by BLK and IN forward until the next " character is encountered.

XMTV (--)

Starting from the location in virtual memory pointed to by BLK and IN, outputs characters through "EMIT until a " character is encountered, which it does not output.

XCOUNT   (adr -- adr+2 xcount)

Extracts a two-byte count from an extended string, and leaves the count on top of the address + 2.

M: xxx, (blk in --)
     xxx: (--)

Generally used after V". Takes a virtual memory pointer from the stack, and creates a word xxx which when executed will push the virtual memory pointer to BLK and IN and then exectue XMTV, thus retrieving a message from disk. See Strolling... for an example.

V: xxx, (blk in --)
     xxx: (--)

Creates a word xxx which when executed pushes the virtual memory pointer which was on stack at the time of its creation to BLK and IN.

V$TP   (-- XCOUNT)

Extracts a two-byte string count from the disk location to which BLK and IN point. leaves it on stack, and bumps the virtual memory pointer made up of BLK and IN twice.

V$@   (-- X$=PAD)

Extracts the extended string in virtual memory pointed to by BLK and IN. The string is left at PAD.

V$*EMT   (--)

Sends the extended string pointed to by BLK and IN through *EMIT.

V$!   (X$ --)

Stores the extended string on stack to virtual memory starting at the location pointed to by BLK and IN.

X"   (---X$=PAD)

Reads the following characters until the delimeter " as an extended string and stores the string at PAD. Operates from screens only. Crosses block and screen boundaries without additional code. Do not use --> to cross screens, as --> will just become part of the string.

ALTSAV   (--)

Copies variables BLK and IN to quans ALTBLK and ALTIN respectively.

ALTREC   (--)

Copies quans ALTBLK and ALTIN to variable BLK and IN respectively.

ALTINIT (scr --)

Sets up ALTBLK and ALTIN to point to screen scr. ALTBLK and ALTIN form an auxiliary virtual memory pointer that is used to keep track of how far messages have been compiled onto the destination disk.

ALT$! (X$ --)

Like V$! except stores string through alternate virtual memory pointers made up of ALTBLK and ALTIN.

LIII. TEXT COMPRESSION AND AUTO TEXT FORMATTING

SUPPLIED SOURCE LISTING

```
Screen:    1
```

```
Screen:    2
```

```
Screen:    3
```

```
Screen:    4
( Transients:  setup               )
'( QUAN )( 5 KLOAD )

BASE @ DCX

HERE

741 @ 4000 - DP !
( SUGGESTED PLACEMENT OF TAREA )


HERE CONSTANT TAREA
QUAN TP
QUAN TPFLAG  1 TO TPFLAG
QUAN OLDDP ( old HERE ) TO OLDDP
                                -->
```

```
Screen:    5
( Xsients: TRANSIENT PERMANENT )
( Expanded from code by Phillip)
( Wasson, in Forth Dimensions  )

: TRANSIENT                 ( -- )
  TPFLAG NOT
  IF HERE TO OLDDP TP DP !
     1 TO TPFLAG
  ENDIF ;

: PERMANENT                 ( -- )
  TPFLAG
  IF HERE TO TP OLDDP DP !
     0 TO TPFLAG
  ENDIF ;
                               -->
```

```
Screen:    6
( Transients:  DISPOSE             )
: DISPOSE   PERMANENT
  CR ." Disposing..." VOC-LINK
  BEGIN DUP     0 53279 C!
   BEGIN @ DUP TAREA U<
   UNTIL DUP ROT ! DUP 0=
  UNTIL DROP VOC-LINK @
  BEGIN DUP 4 -
   BEGIN DUP     0 53279 C!
    BEGIN PFA LFA @ DUP TAREA U<
    UNTIL
     DUP ROT PFA LFA ! DUP 0=
   UNTIL DROP @ DUP 0=
  UNTIL DROP [COMPILE] FORTH
  DEFINITIONS ." Done" CR ;
  PERMANENT                BASE !
```

```
Screen:   7
```

```
Screen:   8
```

```
Screen:   9
```

```
Screen:  10
( Quan:  ASSIGN                               )



'( CFALIT
: ASSIGN [COMPILE] CFALIT ;
IMMEDIATE --> )(  )

: ASSIGN                         ( -- cfa )
  STATE @
  [COMPILE] [
  [COMPILE] ' CFA SWAP
  IF ]
  ENDIF [COMPILE] LITERAL ;
  IMMEDIATE
                                            -->
```

```
Screen:  11
( Quan:  TO  AT                               )

: TO
  -FIND 0= 0 ?ERROR DROP
  STATE @
  IF ,
  ELSE EXECUTE
  ENDIF ; IMMEDIATE

: AT
  -FIND 0= 0 ?ERROR DROP
  2+ STATE @
  IF ,
  ELSE EXECUTE
  ENDIF ; IMMEDIATE
  ( corrected )                             -->
```

```
Screen:  12
( Quan:  [206]  [2!4]                         )

ASSEMBLER HEX

LABEL (206)
  A0 C, 06 C, B1 C, W  C, 48 C,
  C8 C, B1 C, W  C, 4C C, PUSH ,

LABEL (2!4)
  A0 C, 04 C, B5 C, 00 C, 91 C,
  W  C, C8 C, B5 C, 01 C, 91 C,
  W  C, 4C C, POP ,




                                            -->
```

```
Screen:  13
  0 ( Quan:  [2V6]                    )
  1
  2 LABEL (2V6)
  3   A0 C, 07 C, B1 C, W  C, 48 C,
  4   88 C, B1 C, W  C, 85 C, W  C,
  5   68 C, 85 C, W 1+ C,
  6   A0 C, 00 C, 4C C, W 1- ,
  7
  8
  9
 10
 11
 12
 13
 14
 15                                -->
```

```
Screen:  14
  0 ( Quan:  patch for CREATE         )
  1
  2 DCX
  3
  4 : (PTCH)               ( system )
  5   SWAP >R R = 251 R = 249 R> =
  6   OR OR ;
  7
  8 : PTCH                 ( system )
  9   IF [ ' (PTCH) CFA ] LITERAL
 10   ELSE [ ' = CFA ] LITERAL
 11   ENDIF
 12   [ ' CREATE 63 + ] LITERAL ! ;
 13
 14
 15                                -->
```

```
Screen:  15
  0 ( Quan:  QUAN  VECT                )
  1
  2 : QUAN
  3   ON PTCH LABEL -2 ALLOT
  4   (206) , (2!4) ,
  5   [ ' VARIABLE 4 + ] LITERAL ,
  6   2 ALLOT OFF PTCH ;
  7
  8 : VECT
  9   ON PTCH LABEL -2 ALLOT
 10   (2V6) , (2!4) ,
 11   [ ' VARIABLE 4 + ] LITERAL ,
 12   [ ' NOOP CFA ] LITERAL ,
 13   OFF PTCH ;
 14
 15
```

```
Screen:  16
  0 ( Utils:  UMAX  UMIN  HIDCHR    )
  1
  2 : UMAX             ( u1 u2 -- u3 )
  3   2DUP U<
  4   IF SWAP ENDIF
  5   DROP ;
  6
  7 : UMIN             ( u1 u2 -- u3 )
  8   2DUP U>
  9   IF SWAP ENDIF
 10   DROP ;
 11
 12 '( HIDCHR )(
 13 : HIDCHR
 14   -1 94 ! ; )
 15                                -->
```

```
Screen:  17
  0 ( Utils:  S:  P:                   )
  1
  2 '( S: ;S )(  )
  3 HEX
  4
  5 : S:                      ( f -- )
  6   PFLAG @ SWAP
  7   IF 1 OR ELSE FE AND ENDIF
  8   PFLAG ! ;
  9
 10 : P:                      ( f -- )
 11   PFLAG @ SWAP
 12   IF 2 OR ELSE FD AND ENDIF
 13   PFLAG ! ;
 IA
 15 DCX
```

```
Screen:  18
  0
  1
  2
  3
  4
  5
  6
  7
  8
  9
 10
 11
 12
 13
 14
 15
```

```
Screen:  19
```

```
Screen:  20
( Screen code conversion words )

BASE @ HEX

CODE >BSCD                ( a a n -- )
  A9 C,  03 C,  20 C,  SETUP ,
  HERE  C4 C,  C2 C,  D0 C,  07 C,
  C6 C,  C3 C,  10 C,  03 C,  4C C,
  NEXT ,        B1 C,  C6 C,  48 C,
  29 C,  7F C,  C9 C,  60 C,  B0 C,
  0D C,  C9 C,  20 C,  B0 C,  06 C,
  18 C,  69 C,  40 C,  4C C,  HERE
2 ALLOT 38 C,  E9 C,  20 C,  HERE
SWAP !  91 C,  C4 C,  68 C,  29 C,

                                 -->
```

```
Screen:  21
( Screen code conversion words )

  80 C,  11 C,  C4 C,  91 C,  C4 C,
  C8 C,  D0 C,  D3 C,  E6 C,  C7 C,
  E6 C,  C5 C,  4C C,  ,         C;

CODE BSCD>                ( a a n -- )
  A9 C,  03 C,  20 C,  SETUP ,
  HERE  C4 C,  C2 C,  D0 C,  07 C,
  C6 C,  C3 C,  10 C,  03 C,  4C C,
  NEXT ,        B1 C,  C6 C,  48 C,
  29 C,  7F C,  C9 C,  60 C,  B0 C,
  0D C,  C9 C,  40 C,  B0 C,  06 C,
  18 C,  69 C,  20 C,  4C C,  HERE
2 ALLOT 38 C,  E9 C,  40 C,  HERE
                                 -->
```

```
Screen:  22
( Screen code conversion words )

SWAP !  91 C,  C4 C,  68 C,  29 C,
  80 C,  11 C,  C4 C,  91 C,  C4 C,
  C8 C,  D0 C,  D3 C,  E6 C,  C7 C,
  E6 C,  C5 C,  4C C,  ,


: >SCD   SP@ DUP 1 >BSCD ;
: SCD>   SP@ DUP 1 BSCD> ;






                           BASE !
```

```
Screen:  23
```

```
Screen:  24
```

```
Screen:  25
```

```
Screen:  26
```

```
Screen:  27
```

```
Screen:  28
( AFØ:  quans  vects                 )

QUAN BKGND     ( background chr )
BL TO BKGND
QUAN EOB          ( end of buffer )
QUAN BPTR       ( buffer pointer )
QUAN WWID     ( characters/line )
QUAN B/C      ( bytes/character )
QUAN LWD  ( 1st chr of last wd )
VECT *XMTLN       ( send fmted ln )

LABEL BUF 123 ALLOT       ( buffer )
( Need only be longest line +3 )



                                -->
```

```
Screen:  29
( AFØ:  ?BL  INVBK                   )

: ?BL                      ( -- f )
  C@ 31 AND 0= ;

: INVBK                    ( f -- )
  IF 160
  ELSE BL
  ENDIF TO BKGND ;







                                -->
```

```
Screen:  30
( AFØ:  BUFCLR  BUFINIT              )

: BUFCLR                    ( -- )
  BUF WWID BKGND FILL ;

: BUFINIT                   ( -- )
  WWID BUF + 1- TO EOB
  BUFCLR BUF TO BPTR ;


 38 TO WWID
 BUFINIT

 ( Setup for 0 GR. display )
```

```
Screen:  31
















```

```
Screen:  32
( Justify:  *JUST  LCRH  ^LCHR )

VECT *JUST
QUAN LCHR

: ^LCHR                          ( -- )
   EOB
   BEGIN DUP BUF U> OVER ?BL AND
   WHILE 1-
   REPEAT TO LCHR ;






```

```
Screen:  33
















```

```
Screen:  34
( R,cjust:  ^RJ  ^CJ                    )

'( *JUST )( 16 KLOAD )
: (RCJ)                             ( b -- )
  >R BUF BUF EOB LCHR -
  R> / DUP >R +
  LCHR BUF - 1+ <CMOVE
  BUF R> BKGND FILL ;

: ^RJ                                ( -- )
  1 (RCJ) ;

: ^CJ                                ( -- )
  2 (RCJ) ;

                                        -->
```

```
Screen:  35
( R,cjust:  RGT,LFT,CTRJST              )

: RGTJST                             ( -- )
  ASSIGN ^RJ TO *JUST ;

: LFTJST                             ( -- )
  ASSIGN NOOP TO *JUST ;

: CTRJST                             ( -- )
  ASSIGN ^CJ TO *JUST ;






```

```
Screen:  36
















```

```
Screen:  37
```

```
Screen:  38
( Fjust:  quans  (FPTR)            )

'( *JUST )( 16 KLOAD )

QUAN FDIR   1 TO FDIR
QUAN FPTR
QUAN ?FJ

: (FPTR)                    ( f -- )
  FPTR BUF U( NOT
  FPTR LCHR U> NOT AND ;





                                 -->
```

```
Screen:  39
( Fjust:  FPASS                    )

: FPASS                    ( -- )
  0 TO ?FJ
  BEGIN LCHR EOB U( (FPTR) AND
  WHILE FPTR ?BL
   IF 1 TO ?FJ
      FPTR FPTR 1+ EOB FPTR -
      (CMOVE 1 AT LCHR +!
    BEGIN FDIR AT FPTR +!
     FPTR ?BL NOT (FPTR) NOT OR
    UNTIL
   ENDIF FDIR AT FPTR +!
  REPEAT ;

                                 -->
```

```
Screen:  40
( Fjust:  ^FJ  FILJST              )

: ^FJ                           ( -- )
  1 TO ?FJ
  BEGIN LCHR EOB U( ?FJ AND
  WHILE FDIR 0>
   IF BUF ELSE LCHR ENDIF
   TO FPTR FPASS
  REPEAT FDIR MINUS TO FDIR ;

: FILJST                        ( -- )
  ASSIGN ^FJ TO *JUST ;
```

```
Screen:  41
```

```
Screen:  42
( AFJ:  (LWD)  MOVWD   RETWD       )
'( BKGND )( ;S )

: (LWD)                         ( -- )
  BPTR
  BEGIN 1- DUP BUF U(
        OVER ?BL OR
  UNTIL 1+ TO LWD ;

: MOVWD                         ( -- )
  LWD HERE BPTR LWD - (CMOVE ;

: RETWD                         ( -- )
  HERE BUF BPTR LWD -
  DUP >R CMOVE
  R> BUF + TO BPTR ;             -->
```

```
Screen:  43
( AF1:  [SNDLN]  SENDLN          )

: (SNDLN)                   ( -- )
  '( *JUST ^LCHR *JUST )(   )
  *XMTLN BUFCLR ;

: SENDLN                    ( -- )
  (LWD) LWD BUF U>
  IF MOVWD LWD EOB LWD - 1+
     0 MAX BKGND FILL
  ENDIF (SNDLN)
  LWD BUF >
  IF RETWD
  ELSE BPTR 1- C@ BUF C!
       BUF 1+ TO BPTR
  ENDIF ;                      -->
```

```
Screen:  44
( AF1:  *CR                      )

: *CR                       ( -- )
  BPTR BUF =
  IF BUF WWID BKGND FILL
   '( *JUST
  ELSE ^LCHR )(   )
   '( ^FJ ASSIGN ^FJ )( 0 )
   '( *JUST
   AT *JUST @ <)
   IF *JUST
   ENDIF (   )
  ENDIF
  *XMTLN BUFINIT ;

                               -->
```

```
Screen:  45
( AF1:  *EMIT                    )

: *EMIT                   ( c -- )
  DUP 31 =
  IF DROP *CR
  ELSE '( *TINT *TINT )(   )
      '( *CAP  *CAP  )(   )
      '( *INV  *INV  )(   )
   BPTR C! 1 AT BPTR +!
   BPTR EOB 1+ U>
   IF BPTR 1- ?BL
    IF BPTR EOB 2+ MIN TO BPTR
    ELSE SENDLN
    ENDIF
   ENDIF
  ENDIF ;                      -->
```

```
Screen:  46
( AF1:  *TYPE                    )

: *TYPE          ( addr count -- )
  BEGIN DUP 0>
  WHILE
   OVER C@ 127 AND
  *EMIT 1- SWAP 1+ SWAP
  REPEAT 2DROP ;







                               -->
```

```
Screen:  47
( AF1:  *SPACE[S]  *BACKS        )

: *SPACE                    ( -- )
  BKGND *EMIT ;

: *SPACES                 ( n -- )
  0 MAX -DUP
  IF 0 DO *SPACE LOOP
  ENDIF ;

: *BACKS
  BPTR 1- BUF UMAX TO BPTR
  BL '( *INV *INV )(  )
  BPTR C! ;

                               -->
```

```
Screen:  48
( AF1:  [*."]  *."               )

: (*.")                     ( -- )
  R COUNT DUP 1+
  R> + >R *TYPE ;

: *."
  ASSIGN TYPE ASSIGN (.")
  [ ' ." 13 + ] LITERAL
  ASSIGN (*.") OVER !
  [ ' ." 35 + ] LITERAL
  ASSIGN *TYPE OVER !
  [COMPILE] ."
  (ROT ! ! ;          IMMEDIATE
Screen:  49
















Screen:  50
( Coloring:  *TINT  etc.          )

'( )SCD )( 10 KLOAD )

VECT *TINT

'( CLRBYT )(
0 VARIABLE CLRBYT
: COLOR CLRBYT ! ; )

: ^TINT                     ( c -- c )
   )SCD CLRBYT @
   64 * OR SCD) ;

ASSIGN ^TINT TO *TINT


Screen:  51
















Screen:  52
( Capitalization:  CAP[S]  etc.)

QUAN ?CAP
QUAN ?CAPLK


: CAP                           ( -- )
  1 TO ?CAP ;

: CAPS                        ( f -- )
  DUP TO ?CAPLK TO ?CAP ;

  OFF CAPS


                                  -->

Screen:  53
( Capitalization:  *CAP           )

: *CAP                    ( c -- c )
  ?CAP
  IF
   DUP 127 AND DUP
   122 <= SWAP
   97 >= AND
   IF 32 -
   ENDIF ?CAPLK TO ?CAP
  ENDIF ;






Screen:  54
( Inverse Video:  *INV  etc.    )

QUAN ?INV
VECT *INV

: INVID                       ( f -- )
  128 * TO ?INV ;

: ^INV                    ( c -- c )
  ?INV OR ;

  ASSIGN ^INV TO *INV

  OFF INVID


```

```
Screen:  55
















Screen:  56
 ( Efficient <BUILDS...DOES>     )
 ( Partly after G. B. Lyons )
 --> ( Pick up C, code nxt scr )
 ASSEMBLER  HEX

 LABEL (WIP)
   W )Y LDA,  CLC,   3 # ADC,
   IP STA,  INY,  W  )Y LDA,
   0 # ADC,  IP 1+ STA,
   DEY, RTS,

 LABEL (DOES)
   IP 1+ LDA, PHA, IP LDA, PHA,
   (WIP) JSR, ' VARIABLE 4 +
   JMP,
                DCX               -->

Screen:  57
 ( Efficient <BUILDS...DOES>     )

 ASSEMBLER HEX

 LABEL ^WIP
   B1 C,  W C, 18 C, 69 C, 03 C,
   85 C, IP C, C8 C, B1 C,  W C,
   69 C, 00 C, 85 C, IP 1+ C,
   88 C, 60 C,

 LABEL (DOES)
   A5 C, IP 1+ C, 48 C,
   A5 C, IP C, 48 C, 20 C,
   ^WIP , 4C C, ' VARIABLE 4 + ,

                                  -->

Screen:  58
 ( Efficient <BUILDS...DOES>     )

 : DOES>
   COMPILE (;CODE)
   4C C, (DOES) , ; IMMEDIATE

 : <BUILDS
   CREATE SMUDGE ;

   DCX







Screen:  59
















Screen:  60
 ( Txt comp:  TLABEL               )

 ' ( TRANSIENT )( 2 KLOAD )
 ' ( ^WIP      )( 28 KLOAD )

 TRANSIENT

 : TLABEL                   ( -- )
   HERE TRANSIENT
   CONSTANT PERMANENT
   [COMPILE] ASSEMBLER ;

 : | [COMPILE] FORTH ; IMMEDIATE
 VOCABULARY ^ IMMEDIATE

 PERMANENT                        -->
```

```
Screen:  61
( Txt comp:  DMCP$                     )

HEX

TLABEL DCMP$
  A5 C, IP 1+ C, 48 C,
  A5 C, IP C, 48 C, 20 C, ^WIP ,
  CA C, CA C, 18 C, A5 C, W C,
  69 C, 02 C, 95 C, 00 C,
  A5 C, W 1+ C, 69 C, 00 C,
  95 C, 01 C, A0 C, 01 C,
  C8 C, B1 C, W C,
  10 C, FB C,
  88 C, 98 C,
  A0 C, 00 C, 4C C, PUSH0A ,
                                   -->
```

```
Screen:  62
( Txt comp:  [W=]  [P=]  [S=]  )

TLABEL (W=) ASSEMBLER
  4C C, DCMP$ ,
  ] *TYPE *SPACE ;S [

TLABEL (P=) ASSEMBLER
  4C C, DCMP$ ,
  ] *TYPE ;S [

TLABEL (S=) ASSEMBLER
  4C C, DCMP$ ,
  ] *BACKS *TYPE *SPACE ;S [

DCX
                                   -->
```

```
Screen:  63
( Txt comp:  TC=                       )

TRANSIENT

: TC=
  [COMPILE] ^ DEFINITIONS
  HERE >R TRANSIENT
  <BUILDS [COMPILE] IMMEDIATE
   LATEST C@ 31 AND >R
   LATEST 1+ I' R CMOVE
   R I' + DUP C@ 128 AND SWAP C!
   R> R> 2- , PERMANENT ALLOT
  DOES> @ STATE @
   IF , ELSE EXECUTE ENDIF ; -->


```

```
Screen:  64
( Txt comp:  W=   P=   S=             )

: W=                                ( -- )
  (W=) , CURRENT @
  TC= CURRENT ! ;

: P=                                ( -- )
  (P=) , CURRENT @
  TC= CURRENT ! ;

: S=                                ( -- )
  (S=) , CURRENT @
  TC= CURRENT ! ;

PERMANENT

```

```
Screen:  65
```

```
Screen:  66
```

```
Screen:  67
```

```
Screen:  68
( Numerics:  FMT#                          )

: FMT#                          ( f -- )
  IF
       ASSIGN *TYPE
       ASSIGN *SPACES
       ASSIGN *SPACE
  ELSE
       ASSIGN TYPE
       ASSIGN SPACES
       ASSIGN SPACE
  ENDIF
  [ ' D.   4 + ] LITERAL !
  [ ' D.R 22 + ] LITERAL !
  [ ' D.R 24 + ] LITERAL ! ;
                                          -->
```

```
Screen:  69
( Numerics:  *.   *.R                      )

: *.                            ( n --- )
  ON FMT# . OFF FMT# ;

: *.R                         ( n r --- )
  ON FMT# .R OFF FMT# ;
```

```
Screen:  70
( Typed out:  quans                )

QUAN PRTWID     ( printer ch/ln )
80 TO PRTWID    ( init value    )

QUAN VIDWID     ( video ch/ln   )
38 TO VIDWID    ( init value    )

QUAN PCTR    ( printer line ctr )
0 TO PCTR    ( init value       )

QUAN VIDIND    ( video indent )
0 TO VIDIND    ( init value )
QUAN PRTIND    ( printer indent )
0 TO PRTIND    ( init value )
QUAN PVIND     ( indention )   -->
```

```
Screen:  71
( Typed out:  ?CR PWID ?P,VCR  )

VECT ?CR
QUAN PFLG      ( value for PFLAG )
QUAN PWID
80 TO PWID     ( adjust to suit )

: ?PCR                         ( -- )
  WWID PVIND + PWID >=
  IF CR ENDIF ;

: ?VCR                         ( -- )
  WWID PVIND +
  83 C@ 82 C@ - 1+ <
  IF CR ENDIF ;
                                 -->
```

```
Screen:  72
( Typed out:  PRT: PRINIT VID: )

: PRT:                         ( -- )
  PRTIND TO PVIND
  PRTWID TO WWID
  ASSIGN ?PCR TO ?CR
  2 TO PFLG BUFINIT ;

: PRINIT                       ( -- )
  0 TO PCTR ;

: VID:                         ( -- )
  VIDIND TO PVIND
  VIDWID TO WWID
  ASSIGN ?VCR TO ?CR
  1 TO PFLG BUFINIT ;  VID:  -->
```

```
Screen:  73
( Typed out:  *XMTLNP           )

: *XMTLNP                 ( -- )
  PFLAG @
  PFLG PFLAG !
  BUF WWID PVIND SPACES TYPE
  ?CR PFLG 2 =
  IF 1 AT PCTR +!
     PCTR 60 = ( lines/page )
   IF CR CR CR CR CR CR
      PRINIT
   ENDIF
  ENDIF
  PFLAG ! ;

                           -->
```

```
Screen:  74
( Typed out:  TYPEOUT           )

: TYPEOUT                 ( -- )
  ASSIGN *XMTLNP TO *XMTLN ;

( for buffer fmting, no windows)

  TYPEOUT








                           -->
```

```
Screen:  75
```

```
Screen:  76
```

```
Screen:  77
```

```
Screen:  78
```

```
Screen:  79
```

```
Screen:  80
( Windows:  quans  etc.           )

1 ( )SCD )( 10 KLOAD )

QUAN WADR ( window upleftcr adr )
88 @ 2+ TO WADR  ( crnt. uplft )

QUAN WHGT   ( # lines in window )
24 TO WHGT     ( setup for 0 GR. )

QUAN LPTR   ( wndw line pointer )
0 TO LPTR      ( default to top  )

QUAN B/LN             ( bytes/line )
40 TO B/LN     ( setup for 0 GR. )
                                -->
```

```
Screen:  81
( Windows:  [SCROLL]  SCROLL    )

: (SCROLL)                  ( -- )
  WWID B/C * >R  ( # to cmove )
  B/LN B/C * >R  ( # to advance)
  R WHGT 1- * WADR + WADR
  DO I J + I 4 RPICK CMOVE
  J /LOOP
  WADR WHGT 1- R> * + R>
  BKGND )SCD FILL ;

: ?SCROLL                   ( -- )
  LPTR WHGT =
  IF (SCROLL) -1 AT LPTR +!
  ENDIF ;
                                -->
```

```
Screen:  82
( Windows:  WCLR                  )

: WCLR                    ( f -- )
  B/C WHGT * B/LN * WADR + WADR
  DO I WWID
   BKGND
   )SCD FILL
  B/LN /LOOP
  0 TO LPTR ;







                                -->
```

```
Screen:  83
( Windows:  NAMWND WSTP RECWND )

: NAMWND ( wadr wid hgt b/ch    )
  <BUILDS             ( by$/ln -- )
  , , , , , ;

: WSTP ( wa wid hgt b/c b/l -- )
  TO B/LN TO B/C TO WHGT
  TO WWID TO WADR
  BUF WWID + 1- TO EOB
  WCLR BUFINIT ;

: RECWND                ( system )
  >R -2 8
  DO J I + @
  -2 +LOOP R> DROP WSTP ;     -->
```

```
Screen:  84
( Windows:  *XMTLNW  WINDOUT    )

: *XMTLNW                   ( -- )
  ?SCROLL
  BUF LPTR B/LN * WADR +
  WWID )BSCD 1 AT LPTR +! ;

: WINDOUT                   ( -- )
  ASSIGN *XMTLNW TO *XMTLN ;

  WINDOUT




                                -->
```

```
Screen:  85
```

```
Screen:  86
 ( B&W windows:  BWPRM  BWVCT    )
 '( WADR )( 40 KLOAD )

 : BWPRM   ( col row wid hght -- )
           ( wa wid hgt b/c b/l   )
   ROT 40 * 4 ROLL +
   88 @ + <ROT        ( set up wadr )
   1 40 ;             ( b/chr b/ln )

 : BWVCT                      ( -- )
   '( *TINT ASSIGN NOOP
   TO *TINT )(  )
   '( *INV  ASSIGN ^INV
   TO *INV  )(  ) ;

                                 -->
```

```
Screen:  87
 ( B&W windows:  NAMEBW  MAKEBW )

 : NAMEBW ( col row wid hght -- )
   BWPRM NAMWND
   DOES> RECWND BWVCT ;

 : MAKEBW ( col row wid hght -- )
   BWPRM WSTP BWVCT ;
```

```
Screen:  88
 ( Color windows:  CRPM  CVCT    )
 '( WADR )( 40 KLOAD )

 : CPRM    ( col row wid hght -- )
           ( wa wid hgt b/c b/l   )
   ROT 20 * 4 ROLL +
   88 @ + <ROT        ( set up wadr )
   1 20 ;             ( b/chr b/ln )

 : CVCT                       ( -- )
   '( *TINT ASSIGN ^TINT
   TO *TINT )(  )
   '( <INV  ASSIGN NOOP
   TO *INV  )(  ) ;

                                 -->
```

```
Screen:  89
 ( Color windows:  NAME,MAKECW   )

 : NAMECW ( col row wid hght -- )
   CPRM NAMWND
   DOES> RECWND CVCT ;

 : MAKECW ( col row wid hght -- )
   CPRM WSTP CVCT ;
```

```
Screen:  90
```

```
Screen:  91
  0
  1
  2
  3
  4
  5
  6
  7
  8
  9
 10
 11
 12
 13
 14
 15
```

```
Screen:  92
  0
  1
  2
  3
  4
  5
  6
  7
  8
  9
 10
 11
 12
 13
 14
 15
```

```
Screen:  93
  0
  1
  2
  3
  4
  5
  6
  7
  8
  9
 10
 11
 12
 13
 14
 15
```

```
Screen:  94
  0
  1
  2
  3
  4
  5
  6
  7
  8
  9
 10
 11
 12
 13
 14
 15
```

```
Screen:  95
  0
  1
  2
  3
  4
  5
  6
  7
  8
  9
 10
 11
 12
 13
 14
 15
```

```
Screen:  96
  0
  1
  2
  3
  4
  5
  6
  7
  8
  9
 10
 11
 12
 13
 14
 15
```

Screen: 97

Screen: 98

Screen: 99

Screen: 100
```
( Vrtxt:  VRTADJ  VRTCX                )

: VRTADJ                        ( -- )
  ?LOADING
  IN @ B/BUF >=
  IF 0 IN ! 1 BLK +!
  ENDIF ;

: VRTCX                      ( -- adr )
  VRTADJ
  BLK @ BLOCK IN @ + ;




                                   -->
```

Screen: 101
```
( Vrtxt:  VRTC@,!  VRTSAV,REC  )

: VRTC@                       ( -- b )
  VRTCX C@ ;

: VRTC!                       ( b -- )
  VRTCX C! UPDATE ;

QUAN OBLK QUAN OIN

: VRTSAV                ( -- blk in )
  BLK @ TO OBLK  IN @ TO OIN ;

: VRTREC                ( blk in -- )
  OIN IN ! OBLK BLK ! ;
                                   -->
```

Screen: 102
```
( Vrtxt:  NXTVRT  RELVRT               )

: NXTVRT                        ( -- )
  1 IN +! VRTADJ ;

: RELVRT                ( offset -- )
  ?LOADING
  IN @ + B/BUF /MOD BLK +!
  DUP 0<
  IF B/BUF + -1 BLK +!
  ENDIF IN ! ;




                                   -->
```

```
Screen: 103
( Vrtxt:  V"  XMTV  XCOUNT      )

: V"                     ( -- blk in )
  VRTADJ BLK @ IN @
  BEGIN VRTC@ 34 = NXTVRT
  UNTIL ;

: XMTV                          ( -- )
  BEGIN VRTC@ DUP 34 <>
  WHILE *EMIT NXTVRT
  REPEAT NXTVRT DROP ;

: XCOUNT      ( adr -- adr+2 cnt )
  DUP @ SWAP 2+ SWAP ;


                                 -->
```

```
Screen: 104
( Vrtxt:  M:  V:                )

: M:                     ( blk in -- )
  <BUILDS , ,
  DOES> VRTSAV
  DUP @ IN !
  2+ @ BLK !
  XMTV *CR VRTREC ;

: V:                     ( blk in -- )
  <BUILDS , ,
  DOES>
  DUP @ IN !
  2+ @ BLK ! ;


                                 -->
```

```
Screen: 105
( Vrtxt:  EN,DECRYPT  example  )

-->

: ENCRYPT               ( c1 -- c2 )
   117 - DUP 0<
   IF 256 + ENDIF ;

: DECRYPT               ( c2 -- c1 )
   117 + DUP 255 >
   IF 256 - ENDIF ;





                                 -->
```

```
Screen: 106
( Vrtxt:  V$TP  V$@  V$*EMT     )

: V$TP                   ( -- XCOUNT )
  VRTC@ ( DECRYPT ) NXTVRT VRTC@
  ( DECRYPT ) NXTVRT 256 * + ;

: V$@                ( -- X$ [=PAD] )
  PAD 2+ V$TP DUP PAD ! 0
  DO VRTC@ OVER C! 1+ NXTVRT
  LOOP DROP PAD ;

: V$*EMT                        ( -- )
  V$TP 0
  DO VRTC@ ( DECRYPT )
      *EMIT NXTVRT
  LOOP ;                         -->
```

```
Screen: 107
( Vrtxt:  V$!                   )

: V$!                       ( X$ -- )
  DUP @ 2+ 0
  DO DUP C@ ( ENCRYPT )
      VRTC! 1+ NXTVRT
  LOOP DROP ;









                                 -->
```

```
Screen: 108
( Vrtxt:  X"                    )

: X"                 ( -- X$ [=PAD] )
  0 PAD ! PAD 2+
  BEGIN VRTC@ DUP 34 <>
  WHILE OVER C! 1+
          1 PAD +! NXTVRT
  REPEAT NXTVRT 2DROP PAD ;








                                 -->
```

```
Screen: 109
( Vrtxt:                               )















                                     -->
```

```
Screen: 110
( Vrtxt :  ALTSAV,REC                  )

QUAN ALTBLK  QUAN ALTIN

: ALTSAV                       ( -- )
  BLK @ TO ALTBLK
  IN @ TO ALTIN ;

: ALTREC                       ( -- )
  ALTBLK BLK !
  ALTIN  IN  ! ;





                                     -->
```

```
Screen: 111
( Vrtxt:  ALTINIT SCRDSK DSTDSK)

: ALTINIT             ( screen -- )
  B/SCR * TO ALTBLK
  0 TO ALTIN ;


: SRCDSK                       ( -- )
  CR ." Insert source disk and p
ress START." WAIT CR ;

: DSTDSK                       ( -- )
  CR ." Insert dest. disk and pr
ess START." WAIT CR ;

                                     -->
```

```
Screen: 112
( Vrtxt:  ALT$!  MSG:                  )

: ALT$!                        ( X$ -- )
  VRTSAV ALTREC V$!
  ALTSAV VRTREC ;

: MSG:                         ( X$ -- )
  ( $ENCRYPT )
  <BUILDS ( DR1 or ) DSTDSK
   ALTBLK , ALTIN , ALT$!
   FLUSH ( DR0 or ) SRCDSK
  DOES> VRTSAV
   DUP @ SWAP 2+ @ IN ! BLK !
   V$*EMT ( or )
    ( V$@ $DECRYPT XCOUNT *TYPE )
   VRTREC ;
```

```
Screen: 113
```

```
Screen: 114
```

Screen: 115

```
















```

Screen: 116

```
( For demos:  UMOVE  $!           )

*( $! ;S )(  )

: UMOVE                     ( a a n -- )
  <ROT OVER OVER U<
  IF
    ROT <CMOVE
  ELSE
    ROT CMOVE
  ENDIF ;

: $!
  OVER C@ 1+ UMOVE ;

                                   -->
```

Screen: 117

```
( For demos:  ["]  "               )

: (")                          ( -- $ )
  R DUP C@ 1+ R> + >R ;

: "
  34 ( Ascii quote )
  STATE @
  IF                        ( cccc" -- )
    COMPILE (") WORD
    HERE C@ 1+ ALLOT
  ELSE
    WORD HERE         ( cccc" -- $ )
    PAD $! PAD
  ENDIF ;
  IMMEDIATE
```

Screen: 118

```
















```

Screen: 119

```
















```

Screen: 120

```
( X" ... " demo                    )

X" When you are going to take in
 hand any act, remind yourself w
hat kind of an act it is.  If yo
u are going to bathe, place befo
re yourself what happens in the
bath: some splashing the water,
 others pushing against one anot
her, others abusing one another,
 and some stealing:  and thus wi
th more safety you will undertak
e the matter, if you say to your
self, I now intend to bathe, and
 to maintain my will in a manner
 comformable to nature.  And so
```

```
Screen: 121
  0 you will do in every act:  for t
  1 hus if any hindrance to bathing
  2 shall happen, let this thought b
  3 e ready:  it was not this only t
  4 hat I intended, but I intended a
  5 lso to maintain my will in a way
  6  conformable to nature; but I sh
  7 al not maintain it so, if I am v
  8 exed at what happens.→→Epictetus
  9 , translated by George Long, 187
 10 7.→"
 11
 12 CR
 13 ." The X-quote string is loaded"
 14 CR
 15
```

```
Screen: 122
  0 ( V" ... " M: message-name demo)
  1
  2 V" There is an inconvenience whi
  3 ch attends all abstruse reasonin
  4 g, that it may silence, without
  5 convincing an antagonist, and re
  6 quires the same intense study to
  7  make us sensible of its force,
  8 that was at first requisite for
  9 its invention.  When we leave ou
 10 r closet, and engage in the comm
 11 on affairs of life, its conclusi
 12 ons seem to vanish, like the pha
 13 ntoms of the night on the appear
 14 ance of the morning; and 'tis di
 15 fficult for us to retain even th
```

```
Screen: 123
  0 at conviction, which we had atta
  1 in'd with difficulty.  This is s
  2 till more conspicuous in a long
  3 chain of reasoning, where we mus
  4 t preserve to the end the eviden
  5 ce of the first propositions, an
  6 d where we often lose sight of a
  7 ll the most receiv'd maxims, eit
  8 her of philosophy or common life
  9 .  I am not, however, without  h
 10 opes...→→David Hume, 1793.→"
 11
 12    M: MSGDEM1
 13 CR
 14 ." MSGDEM1 now exists."
 15 CR
```

```
Screen: 124
  0 ( X" ... " MSG: msg-name demo  )
  1
  2 80 ALTINIT
  3
  4                               X"
  5 'Accessory No. 5 is a pocket com
  6 pass and is used in connections
  7 with putting.  Like suppose for
  8 inst. you land on the green abou
  9 t 10 ft. from the cup, why the n
 10 ext thing is to find out what di
 11 rection the hole is at and this
 12 can't be done and done right wit
 13 hout a compass.→→ At lease I hav
 14 e seen a whole lot of golfers tr
 15 y and putt without no compass, a
```

```
Screen: 125
  0 nd their ball has went from 10 t
  1 o 45 ft. degrees to the right or
  2  left of where the hole is actua
  3 lly located.  This is because th
  4 ey was just guessing where as wi
  5 th a compass they's no guess wor
  6 k about it.  If you miss a putt
  7 with a compass to tell you just
  8 where a hole is at, why it's bec
  9 ause you can't putt so good.'→→R
 10 ing Lardner on New Golf Accesori
 11 es, 1924.→"
 12
 13   MSG:  MSGDEM2
 14 CR ." MSGDEM2 now exists." CR
 15
```

```
Screen: 126
  0 ( More MSG:'s                   )
  1
  2   X" The rat the cat I bought ca
  3 ught escaped.→" MSG: M0
  4
  5 X" There are gold coins here!→"
  6 MSG: M1
  7
  8      X" Aww, gee, Beave!→"
  9      MSG: M2
 10
 11 X" You see, Watson, but you do n
 12 ot observe.→" MSG: M3
 13
 14 X" Never look back; something ma
 15 y be gaining on you.→"        -->
```

```
Screen: 127
( More MSG:'s                      )

MSG: M4

X" 'The precise date at which th
e reversion to cap and gown took
 place, as well as the fact that
 it affected so large a number o
f schools at about the same time
, seems to have been due in some
 measure to a wave of atavistic
sense of comformity and reputabi
lity that passed over the commun
ity at that period.'→→Thorstein
Veblen, 1899.→" MSG: M5

```

```
Screen: 128
```

```
Screen: 129
```

```
Screen: 130
```

```
Screen: 131
```

```
Screen: 132
```

Screen 133 through 161 are blank

```
Screen: 162
```

```
Screen: 163
```

```
Screen: 164
```

```
Screen: 165
```

```
Screen: 166
( Load Chain                              )

    ' ( DUAN  )( 10 LOAD )
     16 LOAD ( utilities )

 ( Do not modify these lines )










                                        -->
```

```
Screen: 167
( Load Chain, options screen    )
28      LOAD ( af, do not modify )
(   34 LOAD ( rgt & ctr justify )
(   38 LOAD        ( fill justify )
(   50 LOAD             ( coloring )
(   52 LOAD     ( capitalization )
(   54 LOAD        ( inverse video )
42      LOAD ( af, do not modify )
(   60 LOAD  ( text compression )
(   68 LOAD ( fmt'd num. output )
(   70 LOAD        ( typed output )
(   86 LOAD ( B&W window output )
(   98 LOAD ( Color wndw output )
  ( select >= 1 of above 3 )
( 100 LOAD ( Virtual mem. text )
( 116 LOAD ( " for demos )
Screen: 168
  Ø
  1
  2
  3
  4
  5
  6
  7
  8
  9
 1Ø
 11
 12
 13
 14
 15
```

```
Screen: 169
  Ø
  1
  2
  3
  4
  5
  6
  7
  8
  9
 1Ø
 11
 12
 13
 14
 15
```

```
Screen: 170
  Ø CONTENTS OF THIS DISK)
  1
  2 LOAD-CHAIN                     166 LOAD
  3
  4 EFFICIENT (BUILDS DOES>
  5 (ALSO LOADED BY IXTEMP)         56 LOAD
  6 QUAN STRUCTURES                 1Ø LOAD
  7 TRANSIENT STRUCTURES             4 LOAD
  8
  9
 1Ø
 11
 12
 13
 14
 15
```

```
Screen: 171
  Ø
  1
  2
  3
  4
  5
  6
  7
  8
  9
 1Ø
 11
 12
 13
 14
 15
```

```
Screen: 172
  Ø
  1
  2
  3
  4
  5
  6
  7
  8
  9
 1Ø
 11
 12
 13
 14
 15
```

```
Screen: 173
  Ø
  1
  2
  3
  4
  5
  6
  7
  8
  9
 1Ø
 11
 12
 13
 14
 15
```

```
Screen: 174
  0
  1
  2
  3
  4
  5
  6
  7
  8
  9
 10
 11
 12
 13
 14
 15
```

```
Screen: 175
  0
  1
  2
  3
  4
  5
  6
  7
  8
  9
 10
 11
 12
 13
 14
 15
```

```
Screen: 176
  0 ( Error messages                              )
  1
  2 Stack empty
  3
  4 Dictionary full
  5
  6 Wrong addressing mode
  7
  8 Is not unique
  9
 10 Value error
 11
 12 Disk address error
 13
 14 Stack full
 15
```

```
Screen: 177
  0 Disk Error!
  1
  2 Dictionary too big
  3
  4
  5
  6
  7
  8
  9
 10
 11
 12
 13
 14
 15
```

```
Screen: 178
  0 ( Error messages                              )
  1
  2 Use only in Definitions
  3
  4 Execution only
  5
  6 Conditionals not paired
  7
  8 Definition not finished
  9
 10 In protected dictionary
 11
 12 Use only when loading
 13
 14 Off current screen
 15
```

```
Screen: 179
  0 Declare VOCABULARY
  1
  2
  3
  4
  5
  6
  7
  8
  9
 10
 11
 12
 13
 14
 15
```

HANDY REFERENCE CARD

valFORTH™ SOFTWARE SYSTEM

Text Compression and Auto Text Formatting

Basic Commands

[illegible]

Text Compression

[illegible]

Typed Output

[illegible]

Windows

[illegible]

Virtual (Disk-based) Memory

[illegible]